Object-Oriented Application Framework for IEEE 1451.1 Standard

Kang B. Lee, *Fellow, IEEE,* and Eugene Y. Song

***Abstract*—The National Institute of Standards and Technology (NIST), Gaithersburg, MD, smart sensor project focuses on the development and support of the IEEE 1451 Smart Transducer Interface Standard for Sensors and Actuators. One of the main objectives of this project is to provide reference implementations and applications of the IEEE 1451 family of standards. This paper describes an object-oriented application framework for the IEEE 1451.1 standard. This framework consists of four layers—the operating system, middleware and tools, the 1451.1 layer, and the application layer. The 1451.1 layer focuses on the class hierarchy of the IEEE 1451.1 standard. It consists of the neutral model and middleware-based 1451.1 model. The application layer focuses on the application system design that is the composition or aggregation among the objects of the 1451.1 application systems. The initial implementation of the object-oriented application framework for IEEE 1451.1 is provided using the Unified Modeling Language (UML) tools and Adaptive Communication Environment (ACE) middleware. The wastewater treatment system is used as an example to initially customize this framework. IEEE 1451.1 application developers can customize their specific applications with the help of this framework, and the time-to-market for 1451.1 applications can be reduced significantly.**

***Index Terms*—IEEE 1451.1, network capable application processor (NCAP), object-oriented application, object-oriented framework, smart transducer, Unified Modeling Language (UML).**

I. Introduction

THE IEEE Instrumentation and Measurement Society's Technical Committee on Sensor Technology has been working to define IEEE 1451, a family of smart transducer interface standards [1]. Fig. 1 shows the IEEE 1451 family of standards. The IEEE 1451.1 Standard for Smart Transducer Interface for Sensors and Actuators—Network Capable Application Processor (NCAP) Information Model, a member of the family, was established to define a common object model and interface specification for the components of a networked smart transducer [2]. Object-oriented development normally includes four stages: analysis, design, implementation, and testing. An object-oriented framework is a technique for reusing these stages in application development within a certain domain or a certain family of problems [3].

A framework is a reusable, "semicomplete" application that can be specified to produce customer applications [4], [5]. An object-oriented framework aims at reusing codes and designs. It provides an important enabling technology for reusing software components [6]. Compared with traditional software system development, object-oriented framework enables higher productivity and shortens the time-to-market of application development. Thus, the object-oriented framework plays a key role in object-oriented development methodology. Unified Modeling Language (UML) is a modeling language for supporting object-oriented modeling and development. It combines the methods of Booch, Rumbaugh, and Jacobson [7]–[9]. The Object Management Group (OMG) accepted UML as its standard for modeling object-oriented systems in 1997 [9]. Hence, UML is a powerful tool for object-oriented modeling and design of complex software system.

The IEEE 1451.1 object model provides an extensible framework for the design and implementation of software for distributed measurement and control systems consisting of networked NCAP's hosting smart transducers [2]. The object model includes a set of hierarchical classes that is a sound foundation for building an application framework. In order to reduce 1451.1 application development time, it is important to have such an object-oriented application framework.

II. Object-Oriented Framework

An object-oriented application framework is a promising technology for verifying proven software designs and implementations in order to reduce the cost and improve the quality of software [5]. The framework concept can be found in the Smalltalk programming environment.[1] The Smalltalk user interface framework, Model-View-Control (MVC), was perhaps the first widely used framework [10]. MacApp is a framework for Macintosh computer applications. It was designed for supporting the implementation of Macintosh applications that consist of one or more windows, one or more documents, and an application object [11]. Although these earlier frameworks are concerned primarily with implementing a standard user interface, frameworks are by no means limited to user interfaces, and they can be defined for many other domains as well, such as operating systems [12] and fire-alarm systems [13]. An object-oriented framework can be used for developing distributed applications [14]. An object-oriented real-time framework can be used for building a distributed real-time control system in robotics and automation [15]. A framework-based approach to real-time development with UML is used for code generation in

Manuscript received June 15, 2004; revised April 26, 2005.

The authors are with the National Institute of Standards and Technology, Gaithersburg, MD 20899-8220 USA (e-mail: kang.lee@nist.gov; ysong@nist.gov).

Digital Object Identifier 10.1109/TIM.2005.851225

[1]Commercial equipment and software, many of which are either registered or trademarked, are identified in order to adequately specify certain procedures. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology (NIST), Gaithersburg, MD, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

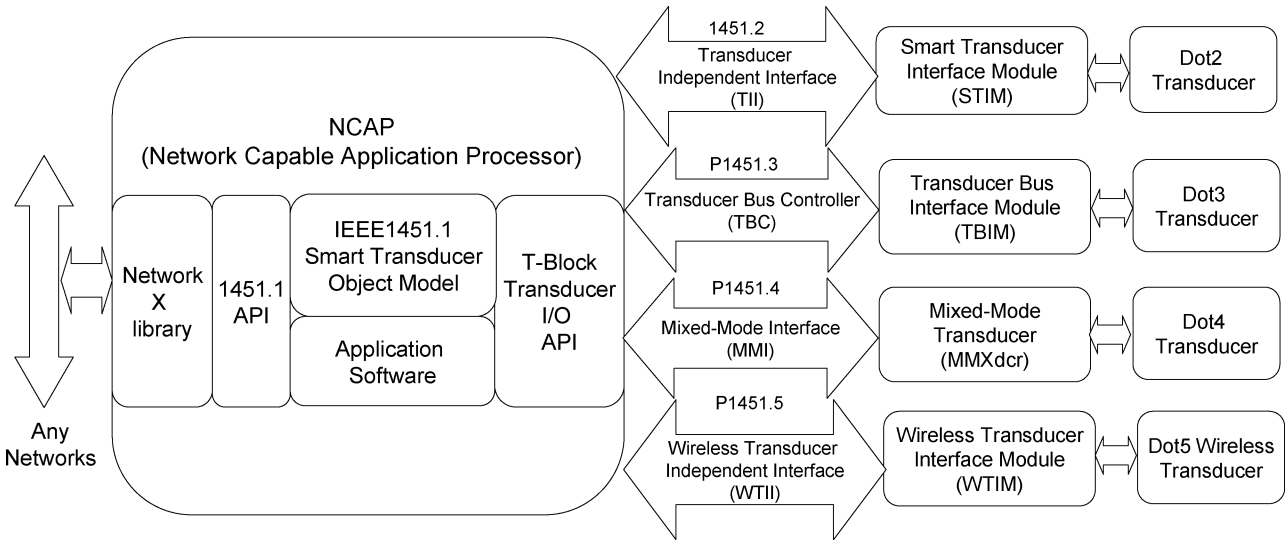

Fig. 1. IEEE 1451 standards family.

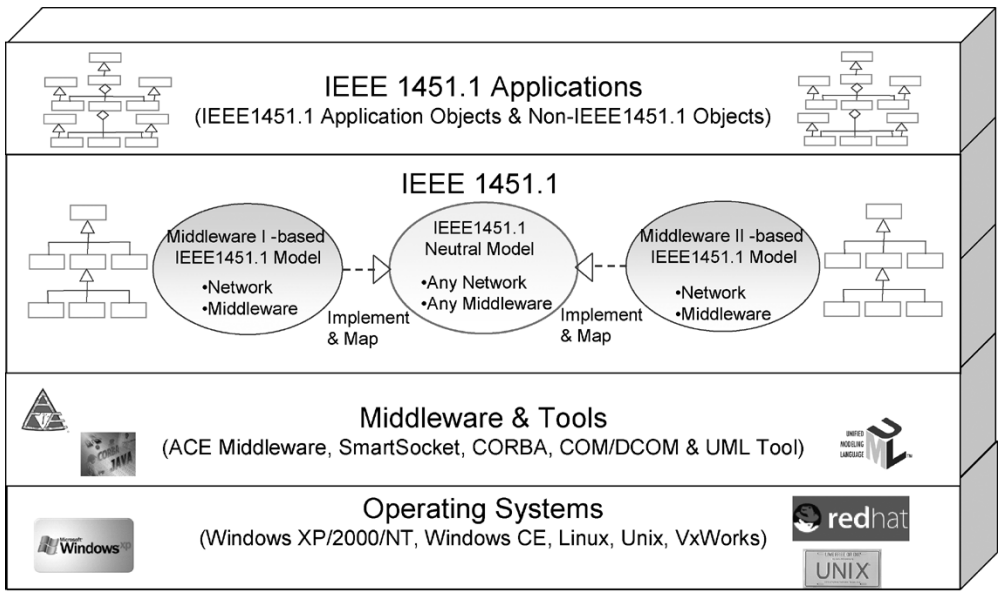

Fig. 2. Layer architecture of object-oriented application framework.

the Rhapsody UML tool [16]. Object-oriented framework can be used for developing distributed manufacturing architectures [17]. Robert and Johnson describes the evolution of a framework as starting from a white-box framework that is reused by subclassing and developing it into a black-box framework that is mostly reused through parameterization [18]. Hayase proposed a three-view model for developing object-oriented frameworks. This three-view model includes a domain analysis view, a layer view, and a mechanism view. The layer view is used to divide a framework into three layers—an infrastructure layer, a generic layer, and a domain layer [19]. Bosch describes a framework consisting of a core framework design and its associated internal increments (if any) with accompanying implementation, such as framework internal increments, and application-specific increments [20]. In summary, object-oriented application frameworks have been widely used in all kinds of domain applications, including real-time measurement and control systems.

III. Object-Oriented Application Framework for IEEE 1451.1

A. Architecture of the Object-Oriented Framework

The IEEE 1451.1 standard defines a common object model for the abstract components of networked smart transducers, together with interface specifications to these components. The object model provides an extensible framework for the design and implementation of software for a distributed measurement and control system. Fig. 2 shows the layer architecture of the object-oriented application framework for the IEEE 1451.1 standard. This framework consists of four layers: operating system layer, middleware and tools layer, 1451.1 layer, and application layer.

- The operating system layer focuses on what platform or operating system the application can be ported to, such as Windows (NT/2000/XP), Linux, Unix, Windows CE, or VxWorks.

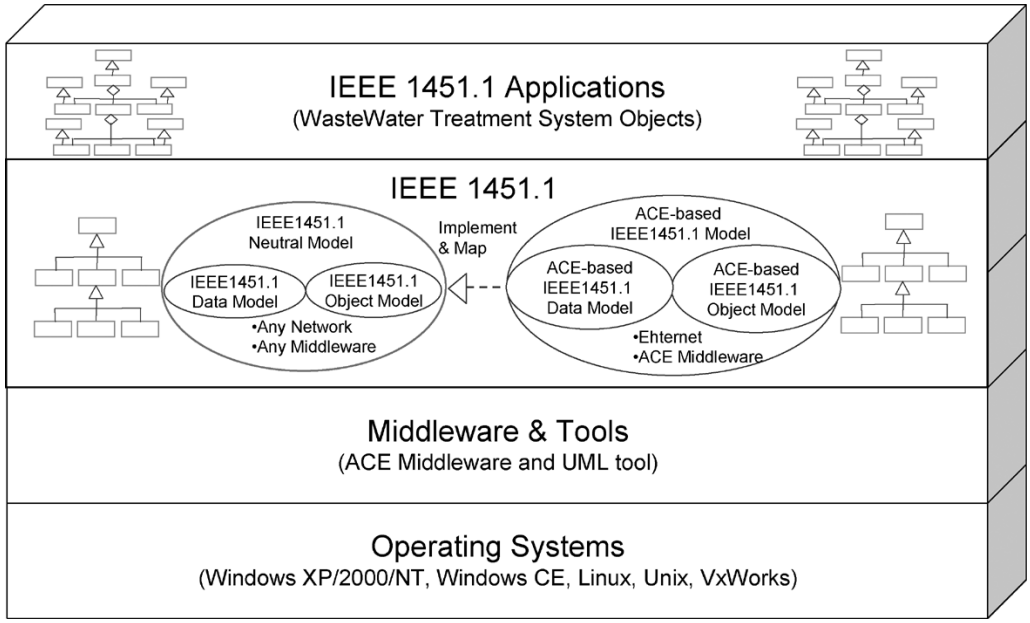

Fig. 3. Customized implementation of object-oriented framework for IEEE 1451.1 standard.

- The middleware and tools layer focuses on what middleware and tools are used to implement the framework and domain applications; for example, the Adaptive Communication Environment (ACE), SmartSocket, Common Object Request Broker Architecture (CORBA) middleware, and all kinds of UML tools.
- The 1451.1 layer focuses on the class hierarchy of 1451.1. This layer consists of the neutral model and middleware-based 1451.1 model.
  - — The neutral model includes operations of all class definitions in the IEEE 1451.1 standard. It is based solely on the 1451.1 standard specifications and is neutral to any network and middleware.
  - — The middleware-based 1451.1 model is the implementation of the neutral model using specific middleware. Each class of middleware-based 1451.1 model should be mapped to the corresponding class of neutral model and implemented using a middleware. Fig. 2 shows the implementation and mapping relationship between the neutral model and middleware-based 1451.1 model.
- The application layer focuses on application system design that is the composition or aggregation among objects of the application systems. This layer consists of the objects of application systems, such as a remote monitoring system, and a distributed measurement and control system.

In Fig. 2, the 1451.1 layer is one kind of white-box framework that relies heavily on static inheritance relationships. It includes the neutral model and middleware-based 1451.1 model. The framework user customizes the framework behavior through subclassing of the framework classes. 1451.1 application developers can use or extend the classes of the neutral model and then implement their specific applications. The middleware-based 1451.1 application developers can inherit the classes of the middleware-based 1451.1 model to implement their specific applications. The application layer is one kind of black-box framework that focuses on dynamic composition or aggregation relationship. It is an application system design that is the composition or aggregation among the objects of the application systems. Application developers can use composition and aggregation relations to combine the application objects or classes that are inherited from the classes of middleware-based 1451.1 models to construct and design their specific applications.

IV. Initial Implementation of Object-Oriented Application Framework Using UML

UML provides many meta-models, such as use-case diagrams and scenarios, object model diagrams, sequence diagrams, collaboration diagrams, charts, component diagrams, and deployment diagrams using an object-oriented approach. We use the object model diagram and state chart of UML to represent 1451.1 classes and class hierarchy and state machines of 1451.1 objects. The object-oriented framework for the 1451.1 standard includes a white-box framework (1451.1 layer) and a black-box framework (application layer). The object-oriented framework was initially implemented based on the ACE middleware and Ethernet using UML tools. The wastewater treatment system was used as an example to customize the framework. Fig. 3 shows a customized implementation of the object-oriented application framework for the IEEE 1451.1 standard.

A. IEEE 1451.1 Layer

The 1451.1 standard defines the software architecture via three models—object model, data model, and communication model. The object model specifies the software component types used to design and implement application systems and provides software building blocks for application systems. The data model specifies the types and forms of information com-

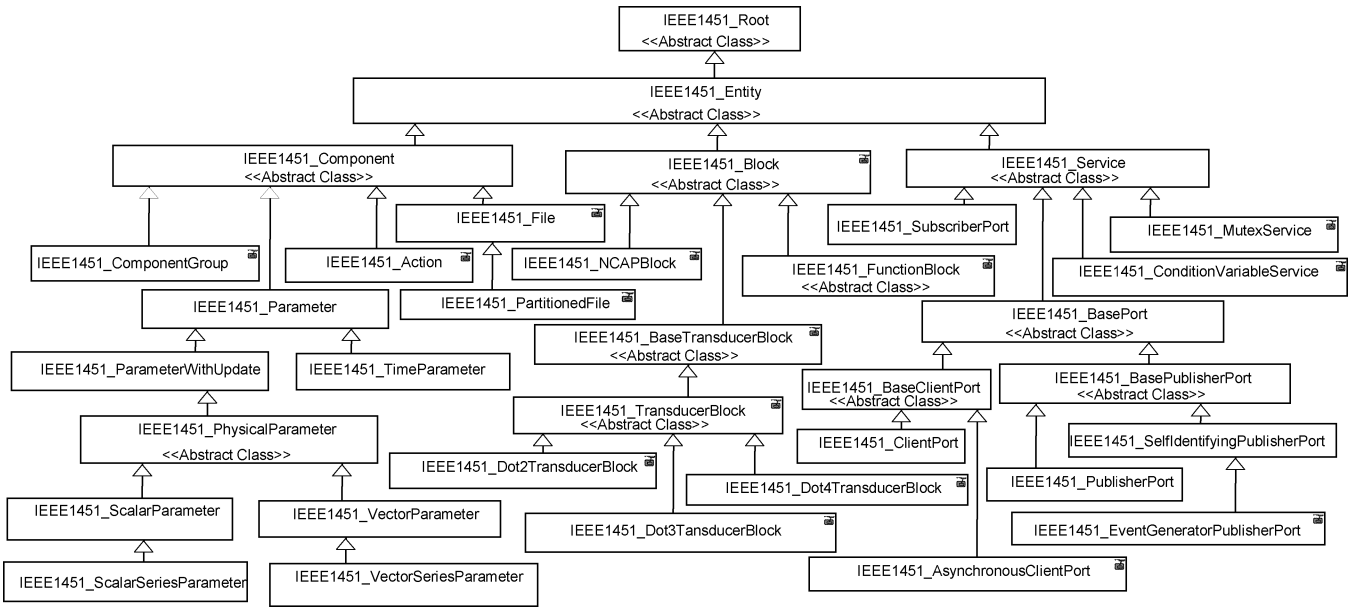

Fig. 4. Class hierarchy of IEEE 1451.1 object model.

municated across 1451.1-specified object interfaces for both local and remote communications. The 1451.1 layer includes the neutral model and middleware-based 1451.1 model that rely heavily on static inheritance relationships.

1) IEEE 1451.1 Neutral Model Using UML: The IEEE 1451.1 neutral model is independent of any networks and middleware. Application developers can extend this neutral model to implement their specific 1451.1 applications. The neutral model has been successfully compiled, linked, and established as a static and dynamic library using UML tools. The neutral model consists of a data model and object model.

a) Data model of IEEE 1451.1 neutral model: The 1451.1 data types can be classified into primitive data type and derived data type (or structured data type) [2]. The primitive data types include Boolean, integers (8, 16, 32, and 64 bits and unsigned), floats (32 and 64 bits), and Octet (an 8-bit unsigned char). The primitive data types can be mapped into C++ data types; for example, Integers64 in 1451.1 can be mapped into _int64 in C++. The structured data types include data types such as arrays, structs, unions, and enumerations. All kinds of structured data types can be defined as classes except for enumerations. The structured data types can be modeled based on an object-oriented approach using UML tools [21].

b) Object model of IEEE 1451.1 neutral model: The object model specifies the software component types used to design and implement application systems. In UML, a class diagram specifies the system structure by identifying object classes and their multiplicities, roles, and object relationships. Objects can be represented as classes in UML, the class schema of the standard. The network visible and local operations of the objects can be described as the public operations and protected operations of the class, respectively. All reactive objects have their own state chart. Fig. 4 shows the class hierarchy of the object model. Detail information is included in [21]. The neutral model is neutral to all networks and middleware. Application developers can extend this neutral model and then implement their specific 1451.1 applications. The neutral model will result in higher reuse of software, so with the help of the neutral model, developers can shorten the development time of 1451.1 applications.

2) ACE-Based IEEE 1451.1 Model: ACE is an open-source, object-oriented framework that implements many core patterns for concurrent communication software [22]. ACE can be used as middleware to implement object-orient frameworks. The neutral model can be mapped into the ACE-based 1451.1 model using ACE middleware in order to implement the neutral model. The 1451.1 layer in Fig. 3 shows the mapping between the neutral model and ACE-based 1451.1 model. The mapping includes data model mapping and object model mapping. In this example, the ACE-based 1451.1 model is specific to the Ethernet and ACE middleware and uses ACE objects and design patterns to implement a neutral model (following the arrow). The ACE-based 1451.1 model has been successfully compiled, linked, and established as a static and dynamic library using UML tools.

a) Data model of ACE-based IEEE 1451.1 model: The ACE-based 1451.1 data model can implement the 1451.1 data model using ACE middleware based on the Ethernet. Each data type class of ACE-based data model should map to the correspondent data type of 1451.1 data model and implement it using ACE middleware. The primitive data types can be mapped to the basic data types of the ACE middleware. For example, Float64 in 1451.1 can be mapped to ACE_CDR::Double. The purpose of using ACE primitive data types is to easily marshal and demarshal data transmitted through a network by means of the ACE_CDR class. All kinds of structured data types can be defined as classes except for enumerations based on the data types of ACE middleware. For example, ACE_ObjectDispatchAddress class can be mapped to ObjectDispatchAddress class in 1451.1. An object's ObjectDispatchAddress is independent on both the network used and the 1451.1 implementation.

If the network is Ethernet, then an object's ObjectDispatchAddress could be a triple consisting of: the IP address, a fixed and standard port number, and the object's object tag. Based on

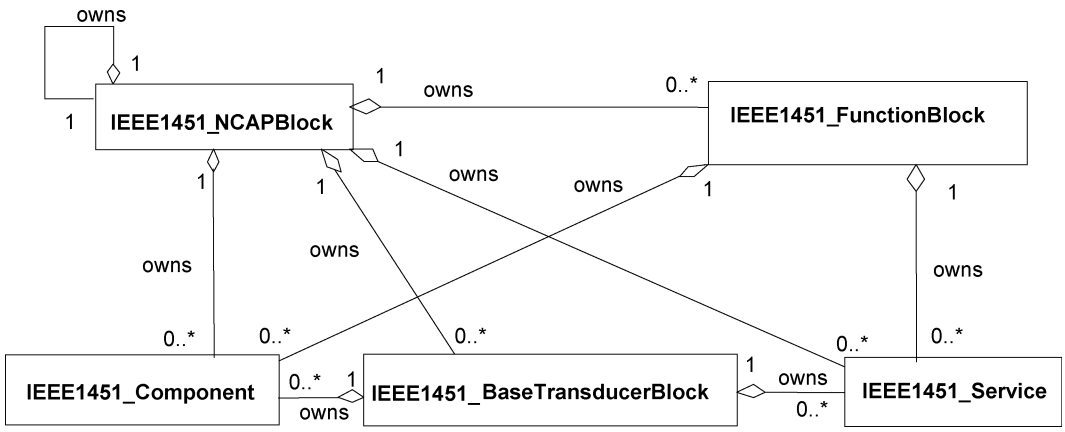

Fig. 5. Dynamic aggregation relationships among IEEE 1451.1 classes.

ACE_INET_Addr class, the ACE_ObjectDispatch-Address can be defined as consisting of iPAddress (ACE_Cstring), portNo (u_short), and objectTag (ACE_ObjectTag).

b) ACE-based IEEE 1451.1 objects: The ACE-based IEEE1451.1 object model is used to implement the object model by means of ACE middleware. A reactor pattern (ACE_Reactor) can be used to do efficient event demultiplexing and dispatching. Active object pattern and acceptor-connector patterns can be used to do client–server communication. ACE_Svc_Handler of ACE is an active class based not only on the ACE_Event_Handler, but also on the ACE_Task that provides the ability to create separate threads and uses message queues to store an incoming data message and to process them concurrently. The ACE-based 1451.1 objects use ACE objects and design patterns to implement corresponding 1451.1 objects. Each class of the ACE-based object model should be mapped or extended from the corresponding class of object model and implemented based on ACE middleware. Compared with the neutral model, all ACE-based 1451 objects are prefixed with ACE_. In the standard, IEEE1451_Entity class is the base class for all network and block services in the IEEE 1451.1 standard. All subclasses of ACE_Entity class can be used as active objects, so these subclasses should inherit the ACE_Svc_Hanlder class of ACE in order to implement client–server and publisher–subscriber communication. The NCAP block is a housekeeper for network communication. ACE_NCAPBlock uses ACE_PublisherPort to publish publications and ACE_SubscriberPort to subscribe to publications. It can use ACE_ClientPort to do client–server communication. ACE_NCAPBlock uses Acceptor–Connector design pattern of ACE to do client–server and publisher–subscriber communication. ACE_ClientPort provides the client-side application interface to client–server communications using operation Execute(). Any subclass of ACE_Entiry can be used as server-side application interface to client–server communication using operation Perform(). The transducer block provides the software connection to the transducer device. The function block provides the transducer application algorithm. The publisher and subscriber port provide publishing and subscribing end points. The mutex service provides application synchronization. Parameters contain the network accessible variables that hold and update the data.

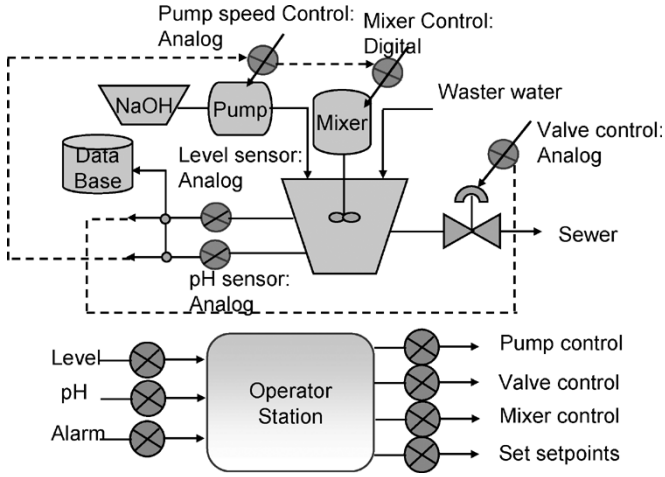

Fig. 6. Wastewater treatment system.

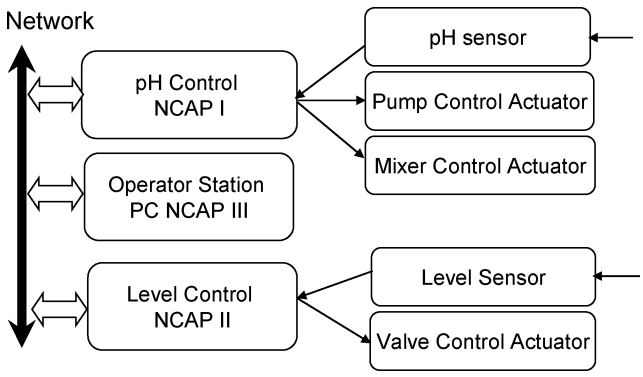

Fig. 7. Components of a wastewater treatment system.

B. Application Layer

An application layer focuses on an application system design that is the composition or aggregation among objects of the application systems. It consists of the objects of the application systems, such as a remote monitoring system and a measurement and control system. Application developers can use aggregation and composition relations to combine the application classes that inherit from the classes of middleware-based 1451.1 models to construct and design their specific applications.

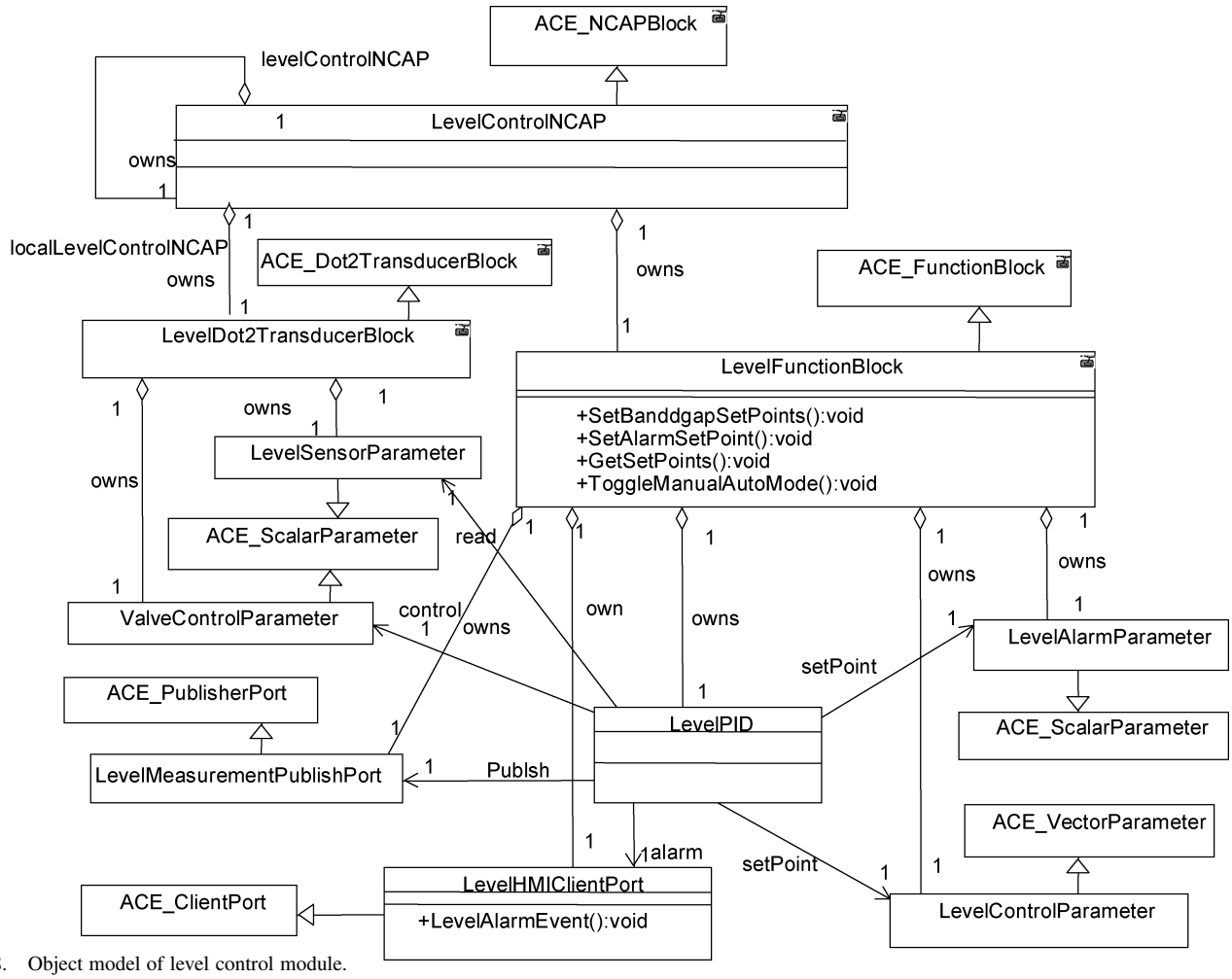

Fig. 8. Object model of level control module.

There are two levels of dynamic ownership relationships among 1451.1 classes shown in Fig. 5 [2]. The first level is the ownership among blocks. The NCAP Block object is a top-level block in the process space. It is the only object type that owns itself (local NCAP block object) and is not owned by any other objects. The NCAP block can own other blocks, such as function block, base transducer block, and local NCAP block. The second level is the ownership between block object and other objects. The ownership relationships of the Block Objects to the Component Objects and Service Objects are shown in the Fig. 5. For example, one NCAP Block can own zero or more component objects and zero or more service objects, respectively. These aggregation relationships in this UML model are very useful in constructing application systems. An NCAP Block object is a logical container of blocks, services, and components. We use the block ownership and inheritance relationships to construct subsystems or modules of a wastewater treatment system, which is shown in Fig. 6.

The wastewater treatment system was taken as an example for customizing the object-oriented framework. The purpose of the wastewater processing system is to monitor the pH value of industrial wastewater and raise it to a predetermined level by mixing in a sodium hydroxide (NaOH) solution before discharging the water into the city sewer. The wastewater treatment system can be divided into three modules: pH control, level control, and operation station module.

Fig. 7 shows the components of the wastewater treatment system. The pH control module is to monitor the pH of industrial wastewater, control the pump speed, and control mixer. The level control adjusts the water level. The operator station displays the pH sensor and level sensor measurement values, logs the data in a database, provides alarms, and manages system control mode (automatic or manual). Each module of wastewater treatment system can be implemented with one NCAP. Each module of the wastewater treatment system can be modeled using the object-oriented approach by means of UML tools and can be customized based on the framework.

Fig. 8 shows the object model of the level control module that consists of one LevelControlNCAP block that owns one LevelFunctionBlock and one LevelDot2TransducerBlock. LevelDot2TransducerBlock, one subclass of ACE_Dot2-TransducerBlock, owns one LevelSensorParameter and one ValveControlParameter. These parameter objects provide an application interface to the level sensor and valve control, respectively. The LevelFunctionBlock, one subclass of ACE_FunctionBlock, owns one LevelPID, one LevelAlarm-Parameter, one LevelControlParameter, one LevelHMIClient-Port, and one LevelMeasurementPublishPort. The LevelPID in the LevelFunctionBlock notifies the operator station if the level

of the wastewater exceeds a given set point. It does so by remotely invoking via client–server communication the LevelAlarmEvent() on the operator station's HMIFunction-Block. The LevelHMIClientPort and LevelMeasurement-PublishPort can be used to communicate with the PHControl module and operator station module. Each class in the level control module can inherit from the corresponding classes in the ACE-based 1451.1 model.

V. Conclusion

This paper describes an object-oriented application framework for the IEEE 1451.1 standard. The neutral model and Adaptive Communication Environment (ACE)-based 1451.1 model developed have been successfully compiled, linked, and established as a static and dynamic library using Unified Modeling Language (UML) tools. The C++ source code of the neutral model is available in the SourceForge website http://open1451.sourceforge.net/. IEEE 1451.1 application developers can extend this neutral model and implement their specific 1451.1 applications using the C++ source code. The customized implementation of the object-oriented application framework for the IEEE 1451.1 standard is provided using UML tools and ACE middleware. The ACE-based 1451.1 application developers can inherit the ACE-based 1451.1 model to implement their specific IEEE 1451.1 applications. The wastewater treatment system has been used as an example to successfully customize this framework. Application developers can use this framework to customize their specific applications, thus reducing the time-to-market for developing 1451.1-based applications.

The authors' future work will add further improvements to the framework and to migrate IEEE 1451.1-based applications into various physical NCAPs in order to verify NCAP functionality.

References

[1] K. Lee, "IEEE 1451: A standard in support of smart transducer networking," in *Proc. Instrumentation Measurement Conf. (IMTC) 2000*, vol. 2, Baltimore, MD, May 1–4, 2000, pp. 525–528.

[2] *A Smart Transducer Interface for Sensors and Actuators-Network Capable Application Processor (NCAP) Information Model*, IEEE Standard 1451.1, Jun. 1999.

[3] T. Ohlsson. Development of object-oriented frameworks. [Online]. Available: http://hem.fyristorg.com/tobias.ohlsson/development_pt97toh.pdf

[4] R. E. Johnson and B. Foote, "Designing reusable classes," *J. Object-Oriented Programming*, pp. 22–35, Jun./Jul. 1988.

[5] M. E. Fayad and D. C. Schmidt, "Object-oriented alication frameworks," *Commun. ACM*, vol. 40, no. 10, pp. 32–38, Oct. 1997.

[6] S. Srinivasan, "Design patterns in object-oriented frameworks," *IEEE Computer*, vol. 32, no. 2, pp. 24–32, Feb. 1999.

[7] M. Fowler and K. Scott, *UML Distilled—A Brief Guide to the Standard Object Modeling Language*, 2nd ed. Reading, MA: Addison-Wesley, 1999.

[8] B. Douglass, *Real-Time UML Model: Developing Efficient Objects for Embedded Systems*, 2nd ed. Reading, MA: Addison-Wesley, 1999.

[9] *http://www.omg.org/uml/* [Online]

[10] A. Goldberg, *Smalltalk-80: The Interactive Programming Environment*, 2nd ed. Reading, MA: Addison-Wesley, 1984.

[11] K. J. Goldberg, *Object-Oriented Programming for Macintosh*. New York: Hayden, 1986.

[12] V. F. Russo, "An object-oriented operating system," Ph.D. dissertation, Univ. of Illinios at Urbana–Champaign, Oct. 1990.

[13] P. Molin and L. Ohlsson, "Points and deviations—A pattern language for fire alarm system," presented at the 3rd Int. Conf. Pattern Language Programming, Monticello, IL, Sep. 1996, Paper 6.

[14] D. C. Schmidt. ASX: An object-oriented framework for developing distributed alications. presented at 6th USENIX C++ Conf. [Online]. Available: http://www.cs.wustl.edu/~schmidt/PDF/C++-USENIX-94.pdf

[15] A. Traub and R. D. Schraft, "Object-oriented realtime framework for distributed control systems," in *Proc. IEEE Int. Conf. Robotics Automation*, vol. 4, Detroit, MI, May 10–15, 1999, pp. 3115–3121.

[16] B. Kadar, L. Monostori, and E. Szelke, "Object-oriented framework for developing distributed manufacturing architectures," *J. Intell. Manuf.*, vol. 9, no. 2, pp. 173–179, Apr. 1998.

[17] M. Fayad, M. E. Fayad, and D. C. Schmidt, *Building Application Frameworks: Object-Oriented Foundations of Framework Design*. New York: Wiley, Sep. 1999.

[18] D. Robert and R. Johnson, "Evolving frameworks: A pattern language for developing object-oriented frameworks," presented at the 3rd Conf. Pattern Language Programming, Montecillio, IL, 1996.

[19] T. Hayase, N. Ikeda, and K. Matsumoto, *A Three-View Model for Developing Object-Oriented Frameworks*. Piscataway, NJ: IEEE Press, 2001.

[20] J. Bosch, P. Molin, P. Mattsson, and M. Bengtsson, *Object-Oriented Frameworks—Problems & Experiences, Object-Oriented Application Frameworks*, M. Fayad, D. Schmidt, and R. Johnson, Eds. New York: Wiley, 1999.

[21] K. Lee and E. Y. Song, "UML model for the IEEE 1451.1 standard," presented at the IEEE Instrumentation Measurement Technology Conf. 2003, Vail, CO, May 2003.

[22] [Online]. Available: http://www.cs.wustl.edu/~schmidt/ACE-overview.html

Kang B. Lee (M'73–SM'00–F'03) received the B.S.E.E. degree from The Johns Hopkins University, Baltimore, MD, in 1971 and the M.S.E.E. degree from the University of Maryland, College Park, in 1981.

He is the Leader of Sensor Development and Application Group of the Manufacturing Engineering Laboratory at the National Institute of Standards and Technology (NIST), Gaithersburg, MD. Since joining NIST, formerly NBS, in 1973, he has worked in the fields of electronic instrumentation design, sensor-based closed-loop machining, robotic manufacturing automation, smart sensor networking, and Internet-based distributed measurement and control systems.

Mr. Lee serves as the Chair of the Instrumentation and Measurement (I&M) Society's Technical Committee on Sensor Technology TC9 that defines the IEEE 1451 standards. He also serves as the I&M Society Delegate to the Sensors Council.

Eugene Y. Song received the Ph.D. degree in manufacturing engineering from Tsinghua University, Beijing, China, in 1998.

He was formerly a Professor at Hebei Institute of Technology, TangShan, China. He is currently a Guest Researcher in the Manufacturing Metrology Division of the Manufacturing Engineering Laboratory at the National Institute of Standards and Technology (NIST), Gaithersburg, MD. He has published 50 journal and conference papers. His current research interests include networked smart sensor interface, objected-oriented modeling based on UML, reference implementation of IEEE 1451 standards, and distributed measurement and control systems.